Q1. Can you create a programme or function that employs both positive and negative indexing? Is there any repercussion if you do so?

ANSWER.

Certainly! Here's an example program that demonstrates the use of both positive and negative indexing in Python:

```python

def get\_characters(s):

positive\_indices = [0, 2, 4]

negative\_indices = [-1, -3, -5]

print("Using positive indexing:")

for i in positive\_indices:

if i < len(s):

print(f"Character at index {i}: {s[i]}")

else:

print(f"Index {i} is out of range")

print("\nUsing negative indexing:")

for i in negative\_indices:

if -i <= len(s):

print(f"Character at index {i}: {s[i]}")

else:

print(f"Index {i} is out of range")

Q2. What is the most effective way of starting with 1,000 elements in a Python list? Assume that all elements should be set to the same value.

ANSWER.

The most effective way to start with a list of 1,000 elements, all set to the same value, is to use list multiplication with a singleton list containing the desired value. This approach leverages the capability of Python to multiply lists, effectively creating a new list with the specified number of elements, each initialized to the same value.

Here's how you can achieve this:

```python

value = 0 # Example value

my\_list = [value] \* 1000

```

Q3. How do you slice a list to get any other part while missing the rest? (For example, suppose you want to make a new list with the elements first, third, fifth, seventh, and so on.)

ANSWER.

To slice a list to extract elements at specific intervals (e.g., every other element), you can use slice notation with a step parameter. Here's how you can achieve this to get elements at odd indices (1st, 3rd, 5th, 7th, and so on):

```python

original\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Slicing with a step of 2 starting from index 0

result\_list = original\_list[::2]

print(result\_list)

```

Q4. Explain the distinctions between indexing and slicing.

ANSWER.

Indexing is used to access individual elements from a sequence, while slicing is used to extract subsequences based on a range of indices. Both operations are fundamental for working with sequences in Python and offer flexibility in accessing and manipulating sequence data.

Q5. What happens if one of the slicing expression's indexes is out of range?

ANSWER.

If one of the slicing expression's indices is out of range (i.e., exceeds the length of the sequence), Python does not raise an error. Instead, it adjusts the index to the nearest valid value within the range of the sequence. This behavior ensures that slicing operations do not result in IndexError and allows for more flexibility when working with sequences.

Q6. If you pass a list to a function, and if you want the function to be able to change the values of the list—so that the list is different after the function returns—what action should you avoid?

ANSWER.

If you want a function to be able to change the values of a list passed to it, you should avoid reassigning the list variable to a new list object within the function.

In Python, lists are mutable objects, which means their values can be modified in place. When you pass a list to a function, you are passing a reference to the list object. Any modifications made to the list object within the function will affect the original list object outside the function.

However, if you reassign the list variable to a new list object within the function, it will create a new local reference to the new list object, and the changes made to this new list object will not affect the original list object passed to the function.

modify the list object passed to the function, rather than reassigning the list variable to a new list object.

Q7. What is the concept of an unbalanced matrix?

ANSWER.

An unbalanced matrix is a matrix that does not have an equal number of rows and columns. It is a rectangular matrix rather than a square matrix, and it may have specific implications for mathematical operations and applications depending on its dimensions and context.

Q8. Why is it necessary to use either list comprehension or a loop to create arbitrarily large matrices?

ANSWER.

List comprehension and loops are necessary for creating arbitrarily large matrices in Python because they provide efficient, dynamic, scalable, readable, and flexible mechanisms for generating matrix elements algorithmically. They enable the creation of matrices of any desired size and structure, making them essential tools for matrix manipulation and numerical computation.